Linux下的汇编程序设计

Linux汇编简介：

一、汇编语言的优缺点：

由于Linux是用C写的，所以C自然而然的就成为了Linux的标准编程语言。大部分人都把汇编给忽略了，甚至在因特网上找资料都是非常的困难，很多问题都需要靠自己来尝试。我认为这样对待汇编语言是不公平的，不能只看到它的缺点，当然也不能只看到它的优点，下面把它的优缺点作一个比较：

优点：汇编语言可以表达非常底层的东西

l 可以直接存取寄存器和I/O

l 编写的代码可以非常精确的被执行

l 可以编写出比一般编译系统高效的代码

l 可以作为不同语言或不同标准的接口

缺点：汇编语言是一个非常低级的语言

l 非常冗长单调，在DOS下编程时就可以体会到

l 易出BUG，且调试困难

l 代码不易维护

l 兼容性不好，与硬件关系非常紧密

总的来说，汇编语言要用在必须的地方，尽量少用汇编编写大型程序，多采用inline模式。

二、汇编语言工具：

DOS下常用的工具MASM和TASM到Linux下就用不起来了，Linux有自己的汇编工具，而且种类非常的多。其中Gas可以算是标准配置，每一种Linux中都包括有Gas，但是GAS采用的不是我们通常在DOS下采用的汇编语法，它采用的是AT＆T的语法格式，与intel语法格式有很大的不同。

如果要采用与DOS接近的语法格式，就必须用另一种汇编工具NASM，NASM基本与MASM相同，但也有不少地方有较大区别，特别涉及到操作系统原理时，与DOS可以说是截然不同。

Linux汇编程序设计：

一、Hello,world!

几乎所有的语言入门篇都是以“Hello,world!”为例，那么我也以Hello,world!为例开始。

;-------------NASM’s standalone Hello-World.asm for Linux --------

section .text

extern puts

global main

main:

push dword msg ;stash the location of msg on the stack.

call puts ;call the "puts" routine (libc?)

add esp, byte 4 ;clean the stack?

ret ;exit.

msg:

db "Hello World!",0

编译：

nasm –f elf hello.asm

gcc –o hello hello.o

说明：这个程序实际上是调用了，Linux系统的puts函数，原理与调用DOS下C语言的函数相同，先用Extern声明puts是外部函数，再把参数（即msg的地址）压入堆栈，最后Call函数实现输出。

我们再来看一个程序：

section .text

global main

main:

mov eax,4 ;4号调用

mov ebx,1 ;ebx送1表示stdout

mov ecx,msg ;字符串的首地址送入ecx

mov edx,14 ;字符串的长度送入edx

int 80h ;输出字串

mov eax,1 ;1号调用

int 80h ;结束

msg:

db "Hello World!",0ah,0dh

（编译同上一个程序）

这个程序与DOS程序十分相似，它用的是linux中的80h中断，相当于DOS下的21h中断，只是因为Linux是32位操作系统，所以采用了EAX、EBX等寄存器。但是Linux作为一个多用户的操作系统与DOS又是有着非常大的区别的。要写出有特色的程序，不了解操作系统和硬件是不行的。下面我介绍一下Linux操作系统。

二、Linux操作系统简介：

操作系统实际是抽象资源操作到具体硬件操作细节之间的接口。对Linux这样的多用户操作系统来说，它需要避免用户对硬件的直接访问，并防止用户之间的互相干扰。所以Linux接管了BIOS调用和端口输入输出，关于端口输入输出方面请参阅Linux IO-Port-Programming HOWTO。而要通过Linux对硬件硬件进行访问就需要用到System Call，实际上是许多C的函数，可以在汇编程序中调用，调用方法与DOS下的汇编完全相同，而且用ASM汇编时不用链接额外的库函数。

Linux与DOS的主要区别在于内存管理、进程（DOS下无进程概念）、文件系统，其中内存管理和进程与汇编编程的关系比较密切：

1、内存管理：

对任一台计算机而言，其内存以及其他资源都是有限的。为了让有限的物理内存满足应用程序对内存的大需求量，Linux采用了称为“虚拟内存”的内存管理方式。Linux将内存划分为容易处理的“内存页”，在系统运行过程中，应用程序对内存的需求大于物理内存时，Linux可将暂时不用的内存页交换到硬盘上，这样，空闲的内存页可以满足应用程序的内存需求，而应用程序却不会注意到内存交换的发生。

2、进程

进程实际是某特定应用程序的一个运行实体。在Linux系统中，能够同时运行多个进程，Linux通过在短的时间间隔内轮流运行这些进程而实现“多任务”。这一短的时间间隔称为“时间片”，让进程轮流运行的方法称为“调度”，完成调度的程序称为调度程序。通过多任务机制，每个迸程可认为只有自己独占计算机，从而简化程序的编写，每个进程有自己单独的地址空间，并且只能由这一进程访问，这样，操作系统避免了进程之间的互相干扰以及“坏”程序对系统可能造成的危害。

为了完成某特定任务，有时需要综合两个程序的功能，例如一个程序输出文本，而另一个程序对文本进行排序。为此，操作系统还提供进程间的通讯机制来帮助完成这样的任务。Linux中常见的进程间通讯机制有信号、管道、共享内存、信号量和套接字等。

三、Linux下的汇编工具：

Linux下的汇编工具可谓百家争鸣，不像DOS下都要给MASM和TASM给控制了。但是Linux下每一种汇编工具都有很大的区别，要想全部掌握几乎是不可能的，下面我介绍几种常用的汇编工具，重点介绍NASM及其使用和语法。

1、GCC

GCC其实是GNU的C语言产品，但它支持Inline Assemble，在GCC中inline assemble使用就像宏一样，但它比宏能更清楚更准确的表达机器的工作状态。

C是汇编编程的一个高度概括，它可以减少许多汇编中的麻烦，特别是在GCC这个C编译器中，assemble似乎起不了多大的作用。

2、GAS

GAS是Linux各版本中基本的汇编工具，但它采用的是AT&T的语法标准与Intel的语法标准有很大的不同，对于DOS编程的我们来说，学习起来是非常困难的。当然如果要精通Linux下的汇编编程，学习GAS也是非常必要的，具体的语法标准可以参看Using GNU Assembler。

3、GASP

GASP是GAS的扩展，它增强了GAS对宏的支持。

4、NASM

NASM是linux中语法与DOS最为相像的一种汇编工具。虽说如此，它与MASM也是有着很大区别的。

l NASM的使用格式如下：

Nasm –f -o

例如：

Nasm -f elf hello.asm

将把hello.asm汇编成ELF object文件，而

Nasm -f bin hello.asm -o hello.com

会把hello.asm汇编成二进制可执行文件hello.com

Nasm –h

将会列出NASM命令行的完整说明。

NASM不会有任何输出，除非有错误发生。

-f 在Linux下主要有aout和ELF两种，如果你不确定你的Linux系统应该用AOUT还是ELF，可以在NASM目录中输入 File nasm ，如果输出nasm: ELF 32-bit LSB executable i386 (386 and up) Version 1表示是ELF，如果输出nasm: Linux/i386 demand-paged executable (QMAGIC)表示是aout。

l NASM与MASM的主要不同：

首先与linux系统一样，nasm是区分大小写的，Hello与hello将是不同的标识符，如果要汇编到DOS或OS/2，需要加入UPPERCASE参数。

其次，nasm中内存操作数都是以[ ]表示。

在MASM中

foo equ 1

bar dw 2

mov ax,foo

mov ax,bar

将被汇编成完全不同的指令，虽然它们在MASM中的表达方式完全一样。而NASM完全避免了这种混乱，它使用的是这样的规则：所有对内存的操作都必须通过[ ]来实现。例如上例中对bar的操作就要写成如下形式 mov ax,[bar]。由此可见，nasm中对offset的使用也是没有必要的（nasm中无offset）。Nasm对[ ]的使用与masm也有所不同，所有的表达式都必须写在[ ]中，下面举两个例子来说明：

Masm Nasm

Mov ax,table[di]

Mov ax,[table+di]

Mov ax,es:[di]

Mov ax,[es:di]

Mov ax,[di]+1

Mov ax,[di+1]

Nasm 中不存储变量类型，原因很简单masm中通过[ ]寻址方式的变量也必须要指定类型。Nasm中不支持LODS, MOVS, STOS, SCAS, CMPS, INS, OUTS，只支持lodsb、lodsw等已经指定类型的操作。Nasm中不再有assume操作，段地址完全取决于存入段寄存器的值。

关于NASM的使用方法及语法还可以参阅NASM使用手册。

结论：

我认为不论是在Windows/DOS下还是在Linux下完完全全用汇编编一个大型程序已经是不可能了，也不会有人愿意去这样做。在windows下我们可以用VC，在Linux/Xwindows下我们可以用C甚至C++ Builder，但是像VC、C++ Builder之类的工具尽量隐藏了底层的调用，同时也阻隔了成为高手的机会，因为编出来的程序无法了解它的执行过程也就使编程中最重要的“可预测”性变得很低。正因为如此汇编才有它存在的必要性，同时还有一个更重要的原因，正如《超级解霸》的作者梁肇新所说：“编程序的重点不是“编”，而是调试程序，理论上的完美在实现的时候会遇到很多细节问题，这些问题必须调试才能解决。我的编程习惯是一天写五天调试，《超级解霸》是调试出来的，而不是写出来的。调试就涉及到汇编的问题，不进行汇编级的调试是不彻底的，也不能让人放心。